Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces

Jennifer Mankoff[1] Scott E. Hudson[2] Gregory D. Abowd[1]

[1]College of Computing & GVU Center
Georgia Institute of Technology
Atlanta, GA 30332-0280, USA
{jmankoff, abowd}@cc.gatech.edu

[2] Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
hudson@cs.cmu.edu

ABSTRACT

Interfaces based on recognition technologies are used extensively in both the commercial and research worlds. But recognizers are still error-prone, and this results in human performance problems, brittle dialogues, and other barriers to acceptance and utility of recognition systems. Interface techniques specialized to recognition systems can help reduce the burden of recognition errors, but building these interfaces depends on knowledge about the ambiguity inherent in recognition. We have extended a user interface toolkit in order to model and to provide structured support for ambiguity at the input event level. This makes it possible to build re-usable interface components for resolving ambiguity and dealing with recognition errors. These interfaces can help to reduce the negative effects of recognition errors. By providing these components at a toolkit level, we make it easier for application writers to provide good support for error handling. Further, with this robust support, we are able to explore new types of interfaces for resolving a more varied range of ambiguity.

Keywords

Recognition-based interfaces, ambiguous input, toolkits, input models, interaction techniques, pen-based interfaces, speech recognition, recognition errors.

INTRODUCTION

Recognition technologies have made great strides in recent years. By providing support for more natural forms of communication, recognition can make computers more accessible. Recognition is particularly useful in settings where a keyboard and mouse are not available, such as very large or very small displays, and mobile and ubiquitous computing.

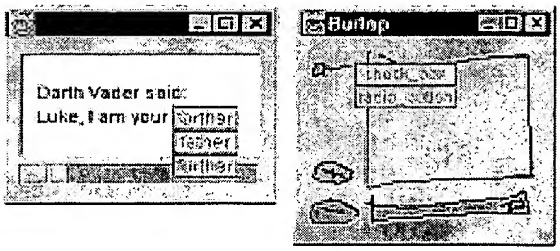
However, systems that make use of recognition technology still labor under a serious burden. Recognizers are error-prone—they make mistakes. This can confuse the user, cause human performance problems, and result in brittle interaction dialogues which can fail due to seemingly small recognition errors. In addition to simple mis-recognition, the inherent ambiguity of language can lead to errors. Ambiguity arises when there is more than one possible way to interpret the user's input. Even a perfect recognizer cannot always eliminate ambiguity, because human language is inherently ambiguous. For example, ambiguity arises in determining the antecedent for words like 'he,' and 'it.'

When humans communicate with computers using speech, gesture, or handwriting, recognition errors can reduce the effectiveness of an otherwise natural input medium. For example, Suhm found that the effective speed of spoken input to computers was only 30 words per minute (wpm) because of recognition errors, even though humans speak at 120 wpm [25]. Similarly, some of the slow success of the PDA market has been attributed to error-prone recognizers.

Although we cannot eliminate ambiguity we can build interfaces which reduce its negative effects on users. For example, Suhm found that user satisfaction and input speed both increased when he added support for multimodal error handling to a speech dictation system [25]. McGee *et al.* found that they were able to reduce ambiguity in a multimodal system by combining results from different recognizers [21]. In the pen and speech communities, the pros and cons of *n*-best lists and repetition [4,26,1,9], as well as secondary input modes (such as soft keyboards) [20,3], have been investigated.

Interfaces which help the user deal with ambiguity depend on having knowledge about the ambiguity resulting from the recognition process. Yet existing user interface toolkits have no way to model ambiguity, much less expose it to the interface components, nor do they provide explicit support for resolving ambiguity. Instead, it is often up to the application developer to gather the information needed by the recognizer, invoke the recognizer, handle the results of recognition, and decide what to do about any ambiguity.

Our primary innovation is to model ambiguity explicitly in the user interface toolkit, thus making ambiguity accessible to interface components. Our toolkit-level solution makes it possible to build re-usable components



(a) (b)

Figure 1: Two examples of a menu of possible interpretations of the user's input. (a) The user is dictating text. (b) The user is drawing a button in Burlap, a user interface sketching tool based on SILK[16].

and strategies for resolving ambiguity. It also allows us to give the user feedback about how their input is being interpreted before we know for sure exactly which interpretation is correct. At the same time, by carefully separating and structuring different tasks within the toolkit, neither application nor interface need necessarily know anything about recognition or ambiguity in order to use recognized input, or to make use of predefined interface components for resolving ambiguity.

Our secondary innovation is to support ambiguity and recognition at the input level, so that objects receive recognition results through the same mechanism as mouse and keyboard events. This allows the application to make use of recognized input without having to deal with it directly or to rewrite any interface components. Our approach differs from other toolkits that support recognized input such as Amulet [17] and various multimodal toolkits [21, 23]. These toolkits provide varying amounts of support for ambiguity, but require the application writer to deal with recognition results directly.

For example, consider a word-prediction application [2]. As the user types each character, the system tries to predict what words may come next. Our toolkit's input system automatically delivers any text generated by the word predictor to the current text focus, just as it would do with keyboard input. The text focus may not even know that a word predictor is involved. If there is more than one possible completion (the text is ambiguous), the toolkit will maintain the relationship between the possible alternatives and any interactive components using that input. The toolkit will automatically pass the ambiguous input to the *mediation* subsystem. This subsystem decides when and how to resolve ambiguity. If necessary, it will involve the user via interfaces that deal with mediation, called *mediators*. For example, it may bring up an *n*-best list, or menu, of alternatives from which the user can select the correct choice (see Figure 1a).

We have extended the user interface toolkit subArctic [8,14] to provide explicit access to ambiguity at an input level, and to provide support for resolving ambiguity through the mediation subsystem. In particular, we have populated the toolkit with some generic mediators that handle a wide variety of types of input, output, and ambiguity.

In order to demonstrate the effectiveness of our ambiguity-aware user interface toolkit, we have built Burlap, a simplified version of the sketch-based user interface builder, SILK (Sketching Interfaces Like Krazy [16]). Burlap, like SILK, allows the user to sketch interactive components (or what we will call *interactors*) such as buttons, scrollbars, and checkboxes on the screen. Figure 1b shows a sample user interface sketched in Burlap. The user can click on the sketched buttons, move the sketched scrollbar, and so on. In this example, the user has drawn a radiobutton, which is easily confused with a checkbox. The system has brought up a default mediator (the same as is shown in Figure 1a), asking the user which interpretation is correct.

The next section describes how ambiguous input is modeled at the toolkit level. We then discuss how ambiguity is handled at the user interface level, using Burlap as a demonstration medium. Burlap illustrates several types of ambiguity, makes use of mediators provided by default in the toolkit, and also demonstrates the benefits of being able to extend these mediators to take advantage of application-specific information. The following section shows how our toolkit-level mechanisms for hiding ambiguity from the application reduce the burden of writing applications that use recognition. We conclude with a description of the variety of recognizers and mediators used in Burlap and some other very simple applications we have built.

MODELING AMBIGUOUS INPUT

Existing user interface toolkits handle input from traditional input sources by breaking it into a stream of autonomous input events. Most toolkits determine a series of eligible objects that may *consume* (use) each event. The toolkit will normally *dispatch* (deliver) an event to each such eligible object in series until the event is consumed. After that, the event is not dispatched to any further objects. Variations on this basic scheme show up in a wide range of user interface toolkits.

In order to extend this model to handle recognized input, we first need to allow the notion of an input source to be more inclusive than in most modern user-interface toolkits. Traditionally only keyboard and mouse (or pen) input is supported. We allow input from any source that can deliver a stream of individual events. In particular, we have made use of input from a speech recognizer (each time text is recognized a new event is generated) and from a context toolkit [24]. The context toolkit handles work of

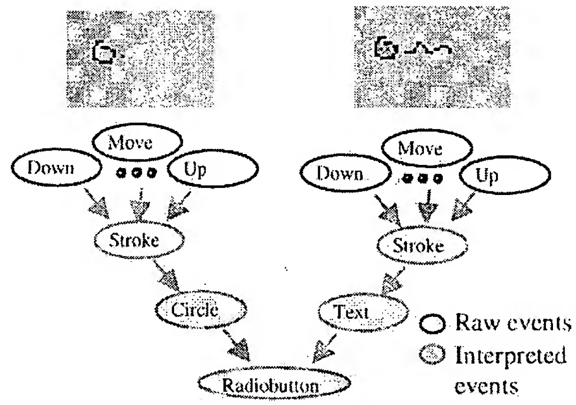


Figure 2: Two mouse strokes and the resulting hierarchical events

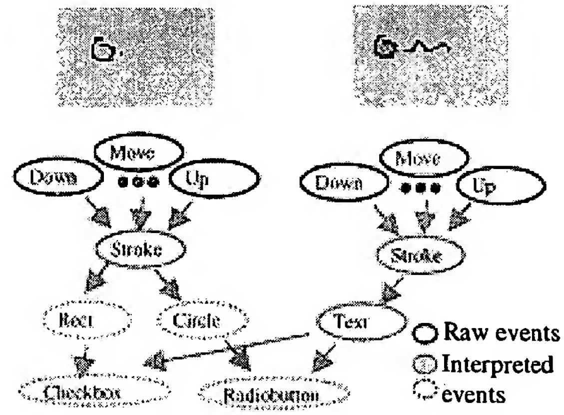


Figure 3: Two mouse strokes and the resulting hierarchical ***ambiguous*** events

gathering information from sensors and interpreting, or abstracting it to provide information such as the identity of users in a certain location (e.g., near the interface).

The next step in supporting recognized input requires us to track explicitly the interpretation and use of input events. This is done using hierarchical events, which have been used in the past in a number of systems (see for example Green's survey [10] and the work by Myers and Kosbie [22]). Hierarchical events contain information about traditional input events (mouse or pen and keyboard) are used, or in the case of recognition, *interpreted*. They can be represented as a directed acyclic graph. Nodes in the graph represent events. Arrows into a node show what events it was derived from. Arrows out of a node show what events were derived from it. Figure 2 shows what this graph looks like when the user draws a radiobutton in Burlap. This graph allows us to explicitly model the relationship between raw events (such as mouse events), intermediate values (such as strokes), derived values (such as recognition results) and what the application does with those values (such as create a radiobutton).

Hierarchical events are generated by requiring each event consumer to return an object containing information about how it interpreted the consumed event [22]. For example, the circle in Figure 2 was created by a gesture recognizer that consumed the stroke the circle derived from and returned the circle as its interpretation of that stroke. Each derived interpretation becomes a new node in the graph. Nodes are dealt with just like raw input events: they are dispatched to consumers, which may derive new interpretations, and so on.

Hierarchical events can also be used to model ambiguity explicitly. Ambiguity arises whenever two events are in conflict (both cannot be correct). For example, the gesture recognizer from Figure 2 may return two possible interpretations (a rectangle and a circle), only one of which can be correct. Figure 3 shows the hierarchical events from Figure 2 with the addition of recognizer-generated ambiguity. The circle, the rectangle, and any interpretation of either are ambiguous.

Just as in the unambiguous case, an ambiguous hierarchy is generated as events are dispatched and consumed. However, unlike most existing systems, we dispatch an event to each consumer in sequence even after the event has been consumed. If an event is consumed by more than one consumer, or a consumer generates multiple interpretations, ambiguity results.

We dispatch every new event (node in the graph) *even if it is ambiguous*. This is critical, because it allows us to put off the process of resolving ambiguity while giving the user early feedback about how events will be used if they are chosen. Remember that ambiguity arises when events are in conflict. We resolve ambiguity by *accepting* one of the possible conflicting events, and *rejecting* the others. The accepted event is correct if it is the interpretation that the user intended the system to make. It is wrong otherwise. It is the job of the mediation subsystem (described in the next section) to resolve ambiguity by deciding which events to accept or reject.

Since we dispatch events even when they are ambiguous, consumers are expected to treat events as tentative, and to provide feedback about them with the expectation that they may be rejected. In particular, we require that feedback related to events be separated from the application actions resulting from those events. By separating feedback about events from action on those events, we can support more flexible strategies for resolving ambiguity. (As described later, a mechanism is also provided for existing interactors which are not ambiguity-aware to operate without this separation, and hence without change).

When ambiguity arises, it is resolved by the mediation subsystem of our extended toolkit. The mediation subsystem consists of a set of objects, called *mediators*, representing different mediation strategies. These

mediator objects, may ignore ambiguity (of types they are not designed to handle), directly resolve ambiguity, or defer resolution of ambiguity until more information is available (or they are forced to make a choice). The ability to defer decisions when appropriate is important to providing robust interaction with ambiguous inputs.

HANDLING AMBIGUITY IN THE INTERFACE

As stated above, the correct interpretation is defined by the user's intentions. It is often appropriate for mediators to resolve ambiguity at the interface level by asking the user which interpretation is correct. For example, the menu of choices shown in Figures 1a and 1b were created by a re-usable mediator that is asking the user to indicate the correct interpretation of their input. This type of mediator is called an *n*-best list, and it is a very common interface technique in both speech and handwriting, as documented in our survey on interfaces for error handling in recognition systems [18]. When the user selects a choice from the list, she is telling the system which of several possible interpretations of her input should be accepted. The system, in turn, informs any object which consumed that input which alternative was accepted.

An *n*-best list is a specific instance of a mediation strategy uncovered by our survey called a *multiple alternative display*. This class of interface techniques gives the user feedback about more than one potential interpretation of her input. In recent work focussing specifically on support for this class of techniques, we identified several dimensions of multiple alternative displays [19] including layout, instantiation time, additional context, interaction, and feedback. For example, in the Pegasus drawing beautification system, user input is recognized as lines. These lines are simply displayed in the location they will appear if selected, rather than converted into a text format and displayed in a menu.

A second common mediation strategy, also discussed in our survey, is *repetition.* A simple type of repetition occurs when automatic mediation results in an error, and the user undoes the error and repeats her input. Alternatively, the user may simply edit some portion of her input. Repetition may take place in the same input mode as the original input, or in a mode with orthogonal or fewer errors. For example, Suhm found that an effective strategy for correcting spoken dictation was to simply edit the resulting text with a pen [25].

One limitation of multiple alternative displays is that they often do not allow the user to specify an interpretation not in the list of alternatives. This can be addressed by combining multiple alternative displays with repetition strategies. Alternatively, when recognition involves graphical objects such as lines or selections, it is enough to simply make the multiple alternative display more interactive. For example, the corner of a selection could be draggable.

Although the examples given above are limited to systems with graphical output, it is also possible to do mediation in other media. For example, we have built a speech I/O *n*-best list style mediator that is equivalent to the mediator in Figure 1b. When the user clicks on an ambiguous button, it asks "Is that a radiobutton?." The user can give one of several answers including "yes" or "it's a checkbox."

Two important design heuristics affect the usability of both types of mediators. The first is to provide sensible defaults. If the top choice is probably right, it should be selected by default so that the user does not have to do extra work. On the other hand, if the recognition process is very ambiguous, such as word prediction, the right default is to do nothing until the user selects an alternative.

The second heuristic is to be lazy. There is no reason to make choices (which may result in errors) until necessary. Later input may provide information that can further disambiguate things. Some input may not need to be disambiguated. Perhaps it is intended solely as a comment or an annotation. For example, in Burlap, there is no need to disambiguate a sketch until the user actually tries to interact with it. Its presence as a drawing on the screen serves a purpose though while ambiguous.

Multiple alternative displays and repetition represent two very general mediation strategies within which there are a huge number of design choices. The decision of when and how to involve the user in mediation can be complex and application-specific. Horvitz's work in decision theory provides a structured mechanism for making decisions about when and how to do mediation [12]. In contrast, our work focuses on supporting a wide range of possible interface strategies.

Modeling Different Types of Ambiguity

The user interface techniques for mediation uncovered in our survey dealt exclusively with one common type of ambiguity, which we call *recognition ambiguity*. However, other types of ambiguity can lead to errors that could be reduced by involving the user. In addition to recognition ambiguity, we often see examples *target ambiguity*, and *segmentation ambiguity*.

Recognition ambiguity results when a recognizer returns more than one possible interpretation of the user's input. For example, Burlap depends upon a recognizer that can turn strokes into interactors. The recognizer is error-prone: If the user draws a rectangle on the screen followed by the symbol for text, the recognizer may mistake the rectangle for a circle (an error), resulting in a radiobutton instead of a checkbox. It is also ambiguous, because it returns multiple guesses in the hope that at least one of them is right and can be selected by mediation. This type of ambiguity is called *recognition ambiguity* (See Figure 1b for an example). In our model of hierarchical ambiguous events, recognition ambiguity is represented by

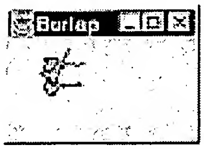


Figure 4—Target ambiguity: Which radiobutton did the user intend to check?

an event with multiple conflicting derivations. Figure 3 shows a case of this.

Target ambiguity arises when the target of the user's input is unclear. For example, the checkmark in Figure 4 crosses two radiobuttons. Which should be selected? A classic example from the world of multimodal computing involves target ambiguity: If the user of a multimodal systems says, *'put that there,'* what does *'that'* and *'there'* refer to? Like recognition ambiguity, target ambiguity results in multiple interpretations being derived from the same source. In the case of recognition ambiguity, one event consumer (a recognizer) generates multiple interpretations. In the case of target ambiguity, the interpretations of an event are generated by multiple potential consumers.

Our third type of ambiguity, segmentation ambiguity, arises when there is more than one possible way to group input events. Did the user really intend the rectangle and squiggle to be grouped as a radiobutton, or was she perhaps intending to draw two separate interactors--a button, and a label? If she draws more than one radiobutton, as in Figure 5, is she intending them to be grouped so that only one can be selected at a time, or to be separate? Should new radiobuttons be added to the original group, or should a new group be created? Similarly, if she writes **'a r o u n d'** does she mean **'a round'** or **'around'**? Most systems provide little or no feedback to users about segmentation ambiguity even though it has a significant effect on the final recognition results.

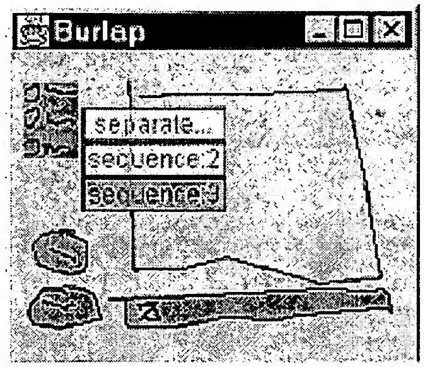


Figure 5--Segmentation ambiguity: The user telling Burlap to group 3 radiobuttons together. When grouped, only one button can be selected at a time.

In the toolkit, segmentation ambiguity corresponds to a situation when two conflicting events are derived from one or more of the same source events. Figure 6 shows the hierarchical events that correspond to the sketch in Figure 5. The user has drawn two radiobuttons on the screen. Just as in Figure 3, these radiobuttons themselves are ambiguous. However, since there are *three* radiobuttons, the interactor recognizer associated with Burlap generates two new interpretations—the first two radiobuttons may be grouped together, or all three may be grouped together. We modified our mediator to add a third possibility—the radiobuttons may all be separate.

Choosing the wrong possibility from a set of ambiguous alternatives causes a recognition error. Many common recognition errors can be traced to one or more of the three types of ambiguity described above. Yet user interfaces for dealing with errors and ambiguity almost exclusively target recognition ambiguity while ignoring segmentation and target ambiguity. For example, of the systems

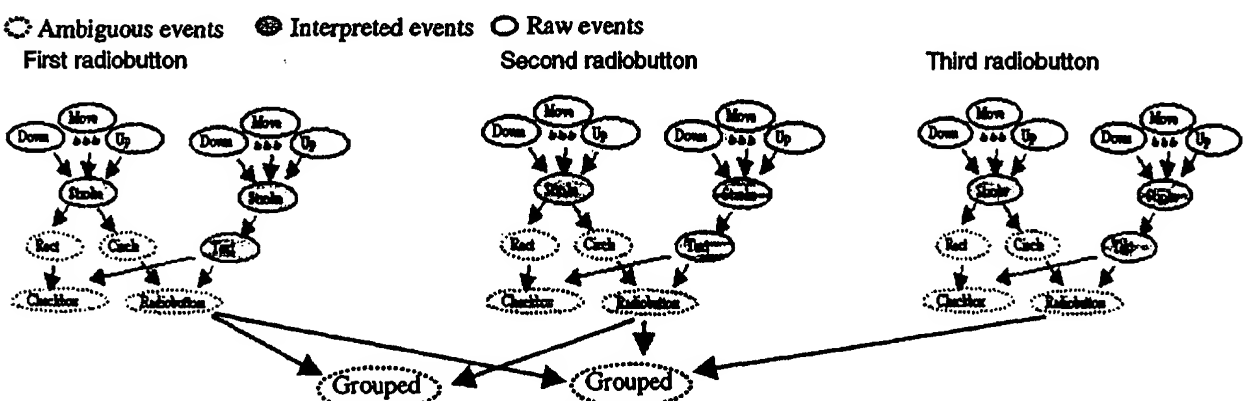


Figure 6: Three radiobuttons resulting in ***Segmentation ambiguity.*** The grayed out portions of the tree are identical to that in Figure 3.

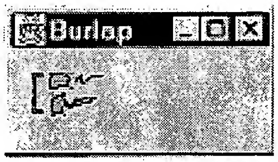


Figure 7: An application—specific mediator asking whether these radiobuttons should be grouped together or separately. If the user draws a slash through the mediator, the buttons will be separated. Any other action will cause them to be grouped (this is the default).

surveyed [18], none dealt directly with segmentation ambiguity. In one case, researchers draw lines on the screen to encourage the user to segment their input appropriately, but they give no dynamic feedback about segmentation [9]. None dealt with target ambiguity.

Extending Mediators

We have extended the concept of mediation. Our re-usable multiple alternative display, shown working with recognition ambiguity resulting from word-prediction and sketches in Figures 1a and 1b, also handles target and segmentation ambiguity (Figure 5).

In addition, we have begun to explore the possibilities for designing better interaction techniques for mediation by making use of application specific information, something that is lacking in many of the most common mediation interfaces found in the literature and in products. For example, most *n*-best lists simply show the alternatives generated by the recognizer, without indicating how different alternatives may affect the application.

Figure 7 demonstrates an application-specific mediator for segmentation ambiguity in Burlap. This mediator handles the same situation as that shown in Figure 5, but it makes use of task-specific knowledge. It knows that radiobuttons drawn near each other are usually intended to be grouped together, so it provides a sensible default: if the user continues to sketch interactors, the mediator will automatically accept the grouping as the correct option. It also knows enough about the task to pick a simple graphical representation of its options rather than trying to pick a word for a graphical relationship without an obvious short verbal description. This also allows it to take up less space and be less obtrusive (since the user probably will pick its top choice anyway). The user need only click on the mediator to reject its suggestion that the radiobuttons be grouped.

Mediation in the Toolkit

Ambiguous events are automatically identified by the toolkit and sent to the mediation subsystem. This system keeps an extensible list of mediators. When an ambiguous event arrives, it passes that event to each mediator in turn until the ambiguity is resolved.

If a mediator is not interested in the event, it simply returns PASS. For example, the *n*-best list mediator in Figure 5 only deals with segmentation ambiguity in radiobuttons—other kinds of ambiguity it simply ignores by returning PASS. If a mediator resolves part of the event graph—by accepting one interpretation and rejecting the rest—it returns RESOLVE. Finally, if a mediator wishes to handle ambiguity, but defer final decision on how it will be resolved, it may return PAUSE. Each mediator is tried in turn until some mediator signifies that it will handle mediation by returning PAUSE or a mediator returns RESOLVE and the complete event graph is no longer ambiguous. The toolkit provides a fallback mediator that will always resolve an ambiguous event (by taking the first choice at each node). This guarantees that every ambiguous event will be mediated in some way.

Not all mediators have user interfaces—sometimes mediation is done automatically. In this case, errors may result (the system may select a different choice than the user would have selected). An example of a mediator without a user interface is the *stroke pauser*. The stroke pauser returns PAUSE for the mouse events associated with partially completed strokes, and caches those events until the mouse button is released. It then allows mediation to continue, and each cached mouse event is passed to the next mediator in the list. Feedback about each stroke or partial stroke still appears on the screen since they are delivered to every interested interactor *before* mediation. However, no actions are taken until the mouse is released (and mediation completed).

Interactive mediators are treated no differently than automatic mediators. The mediators shown in Figure 1b and Figure 5 get events after the *stroke pauser*. When they see an event with ambiguity they know how to handle, they add the menu shown to the interface. They then return PAUSE for that event. Once the user selects an alternative, they accept the corresponding choice; and allow mediation to continue.

The toolkit enforces certain simple guarantees about events that are accepted or rejected. All interpretations of rejected events are also rejected. All sources of accepted events are also accepted. Further, any event that conflicts with an accepted event is rejected.

HIDING AMBIGUITY

We have described an architecture that makes information about ambiguity explicit and accessible. At the same time, we do not wish to make the job of dealing with ambiguity onerous for the application developer. The solution is to be selective about how and when we involve the application. This is possible because we separate the task of mediation from the application and from other parts of the toolkit. Mediation, which happens separately, determines what is accepted or rejected. Note that

mediation can be integrated into the interface even though it is handled separately in the implementation.

It is sometimes too costly to handle the separation of feedback and action. Some things, such as third party software, or large existing interactor libraries, may be hard to rewrite. Also, in certain cases, a rejected event may break user expectations (e.g., a button that depresses yet in the end does nothing).

In order to handle these cases, we provide two simple abstractions. An interactor can be *first-dibs*, in which case it sees events first, and if it consumes an event no one else gets it. Thus ambiguity can never arise. Alternatively, a *last-dibs* interactor only gets an event if no one else wants it, and therefore is not ambiguous. Again, ambiguity will not arise. These abstractions can handle most situations where ambiguity-aware and ambiguity-blind interactors are mixed. This allows conventional interactors (e.g., the buttons, sliders, checkboxes, etc. of typical graphical user interfaces) to function in the toolkit as they always have, and to work along side ambiguity-aware interactors when that is appropriate. This property is important in practice, because we do not wish to force either interface developers, or users, to completely abandon their familiar interfaces in order to make use of recognition technology.

CURRENT STATUS AND FUTURE WORK

We have implemented the infrastructure described in this paper as an extension to the subArctic user interface toolkit. With this extended system, we have built two applications—Burlap, which makes use of pen, speech, and identity of its users as input, and a simple word-prediction text entry system.

In order to build these applications, we have connected an off-the-shelf speech recognizer (IBM's ViaVoice™), a context toolkit [24], and two pen recognizers (A 3rd party unistroke gesture recognizer [6] and an interactor-recognizer based directly on the design used for SILK[16]) to our toolkit. Note that in order to handle speech and identity, we had to add two new event types to the toolkit's input system (which handles mouse and keyboard input only by default). The speech system simply creates "speech" events whenever ViaVoice™ generates some interpretations. The context system creates an identity event each time a piece of relevant identity context arrives. We also used Pegasus, Igarashi's drawing beautification system [15], with a previous version of the extended toolkit, described in [19].

We have populated the toolkit with default mediators for pen (Figure 1a), speech, and identity. These mediators address segmentation and recognition ambiguity. We also built application-specific mediators for segmentation of pen input (Figure 1b) and for identity.

Our work differs from other GUI toolkits that support recognized input such as Artkit [11], and Amulet [17], which require recognizers to return a single, unambiguous result. We track the relationship between input and recognized input using an extended version of hierarchical events [22] that supports ambiguity. Hudson and Newell also present work that tracks ambiguity [13], but their work was not integrated with a toolkit, and confined to individual input handlers rather than supporting ambiguity throughout the input cycle. Multimodal toolkits that provide support for ambiguity do not integrate this within the input handling mechanism of a GUI toolkit and thus lack the flexibility to provide re-usable support for mediation at an interface level [23,21].

In the future, we plan to extend the mediation infrastructure to support fluid negotiation for space on the screen [5]. We will investigate mediators for target ambiguity. We will provide better support for repetition-based mediation, including undo. Once we can undo events, we can allow interpretations to be created after mediation is complete. For example, if additional, relevant information arrives, an accepted alternative may need to be undone and mediation repeated.

CONCLUSIONS

We have provided principled, toolkit-level support for a model of ambiguity in recognition. The purpose of building toolkit-level support for ambiguity is to allow us to build better interfaces to recognition systems. As stated in the introduction, user interface techniques can help reduce the negative effects of ambiguity on user productivity and input speed, among other things. Our solution addresses a wide range of user interface techniques found in the literature, and allows us to experiment with mediators that take advantage of application-specific knowledge to weave mediation more naturally into the flow of the user's interaction.

By making ambiguity explicit, we were able to identify two types of ambiguity not normally dealt with—segmentation and target ambiguity. In addition, because we separate feedback about what *may* be done with user input from action on that input, we can support a much larger range of mediation strategies including very lazy mediation.

Our work is demonstrated through Burlap, for which we have built a series of mediators for segmentation, recognition, and target ambiguity.

ACKNOWLEDGMENTS

We thank Anind Dey for his help with implementation and as a sounding board for these ideas. This work was supported in part by the National Science Foundation under grants IRI-9703384, EIA-9806822, IRI-9500942 and IIS-9800597.

REFERENCES

1. Ainsworth, W.A. and Pratt, S.R. Feedback strategies for error correction in speech recognition systems. *International Journal of Man-Machine Studies*, 39(6), pp. 833-842.
2. Alm, N., Arnott, J.L. and Newell, A.F. Prediction and Conversational momentum in an augmentative communication system. *Communications of the ACM*, 35(6), pp. 833-842.
3. Apple Computer, Inc. The Newton MessagePad
4. Baber, C. and Hone, K.S. Modeling error recovery and repair in automatic speech recognition. *International Journal of Man-Machine Studies*, 39(3), pp. 495-515.
5. Chang, B., Mackinlay, J.D., Zellweger, P. and Igarashi, T. A negotiation architecture for fluid documents. In *Proceedings of UIST'98*, ACM, November 1998, pp.123–132.
6. Christian, A. Long, J.R. Landay, J.A. and Rowe, L.A. Implications for a gesture design tool. In *Proceeding of CHI'99*, May, 1999, pp. 40-47.
7. DragonDictate product Web page. Available at: http://www.dragonsystems.com/products/dragondictate/index.html
8. Edwards, W. K., Hudson, S., Rodenstein, R., Smith, I. and Rodrigues, T. Systematic output modification in a 2D UI Toolkit. In *Proceedings of UIST'97*, ACM, October 1997, pp. 151-158.
9. Goldberg, D. and Goodisman, A. Stylus user interfaces for manipulating text, *in Proceedings of UIST'91*, ACM, 1991, pp.127-135.
10. Green, M. A survey of three dialogue models. *ACM Transactions on Graphics*, 5(3), July, 1986, pp. 244-275.
11. Henry, T.R., Hudson, S.E., and Newell, G.L. Integrating gesture and snapping into a user interface toolkit. In *Proceedings of UIST'90*. October 1990. pp.112-122.
12. Horvitz, E. Principles of mixed-initiative user interfaces. In *Proceeding of CHI'99*, May, 1999, pp. 159-166.
13. Hudson, S., Newell, G. Probabilistic state machines: Dialog management for inputs with uncertainty. In *Proceedings of UIST'92*, ACM, November 1992, pp. 199-208.
14. Hudson, S. and Smith, I. Supporting dynamic downloadable appearances in an extensible user interface toolkit. In *Proceedings of UIST'97*, ACM, October 1997, pp. 159-168.
15. Igarashi, T., Matsuoka, S., Kawachiya, S. and Tanaka, H. Interactive beautification: A technique for rapid geometric design. In *Proceedings of UIST'97*, ACM, October 1997, pp. 105-114.
16. Landay, J. A. and Myers, B.A. Interactive sketching for the early stages of user interface design. In *Proceedings of CHI '95*, 1995, pp.43-50.
17. Landay, J.A. and Myers, B.A. Extending an existing user interface toolkit to support gesture recognition. In *Proceedings of INTERCHI'93*, ACM Press, pp. 91-92.
18. Mankoff, J. and Abowd, G.D. Error correction techniques for handwriting, speech, and other ambiguous or error prone systems. Georgia Tech GVU Center Technical Report, GIT-GVU-99-18, 1999.
19. Mankoff, J. Abowd, G.D. and Hudson, S.E. Interacting with multiple alternatives generated by recognition technologies. Georgia Tech GVU Center Technical Report, GIT-GVU-99-26, 1999.
20. Marx, M. and Schmandt, C. Putting people first: Specifying proper names in speech interfaces. In *Proceedings of UIST'94*, ACM, N.Y. November 1994, pp. 30–37.
21. McGee, D. R., Cohen, P.R., and Oviatt, S. Confirmation in multimodal systems. In *Proceedings of the International Joint Conference of the Association for Computational Linguistics and the International Committee on Computational Linguistics (COLING-ACL '98)*, Montreal, Quebec, Canada.
22. Myers, B.A. and Kosbie, D.S. Reusable hierarchical command objects. In *Proceedings of CHI'96*, 1996, pp. 260-267.
23. Nigay, L. and Coutaz, J. A Generic platform addressing the multimodal challenge. In *Proceedings. of CHI'95*, pp.98-105.
24. Salber, D. Dey, A.K. and Abowd, G.D. The context toolkit: Aiding the development of context-enabled applications. In *Proceeding of CHI'99*, May, 1999, pp. 434-441.
25. Suhm, B. Myers, B. and Waibel, A. Model-based and empirical evaluation of multimodal interactive error correction. In *Proceeding of CHI'99*, May, 1999, pp. 584-591.
26. Zajicek, M. and Hewitt, J. An investigation into the use of error recovery dialogues in a user interface management system for speech recognition. In *Proceedings of IFIP Interact'90*, pp. 755-760.